WHAT IS CLAIMED IS:

1. A method of providing a linearizable multi-compare, single-swap facility for concurrent software, the method comprising:

updating a first application value corresponding to a first targeted location only if the application values corresponding to plural other targeted locations remain unchanged;

snapshotting application values corresponding to the plural other targeted locations; and

employing a pair of single-location synchronizations to that ensure that the application value corresponding to the first targeted location remained unchanged at a linearization point of the snapshot.

2. The method of claim 1,

wherein the first targeted location and at least one of the other targeted locations are non-contiguous.

3. The method of claim 1,

wherein a first one of the single target synchronizations precedes the snapshotting; and

wherein a second one of the single target synchronizations follows the snapshotting.

4. The method of claim 3,

wherein the second one of the single target synchronizations effectuates the updating.

5. The method of claim 1,

wherein the single-location synchronizations retry on failure.

6. The method of claim 1,

wherein the method has a non-blocking property.

7. The method of claim 1,

wherein the non-blocking property includes obstruction-freedom.

8. The method of claim 1,
wherein the single-location synchronizations employ tagged id displacement for ABA avoidance.

9. The method of claim 1, further comprising:
displacing the first application value from the first targeted location prior to the linearization point of the snapshotting.

10. The method of claim 9, wherein the displacing includes:
reading the first application value;
storing the read value in an auxiliary location associated with a tagged id; and
storing, using a first of the single-location synchronizations, the tagged id in the targeted location.

11. The method of claim 9, wherein the displacing is performed by a load-linked sequence that employs one of the single-location synchronizations.

12. The method of claim 1, wherein the snapshotting includes:
collecting (i) application values associated with each of the other targeted locations and (ii) tagged ids, if any, from corresponding tagged id locations until two successive collections indicate identical respective application values and identical respective tagged ids.

13. The method of claim 1, further comprising:
resetting any particular targeted location, including the first targeted location and any of the plural other targeted locations, in connection with retrieval of an associated application value from a corresponding auxiliary location,
the resetting including displacing, using a single-location synchronization, a tagged id stored in the particular targeted location with the associated application value.

14. The method of claim 1,

wherein any particular application value is read either from a corresponding one of the targeted locations, if encoded therein, or from an auxiliary location associated with an id, if instead, the id is encoded therein.

15. The method of claim 1,

wherein the first and other targeted locations are non-contiguous.

16. The method of claim 1,

wherein the single-location synchronizations are compare-and-swap (CAS) synchronizations.

17. The method of claim 1,

wherein the single-location synchronizations are atomic read-modify-write synchronizations.

18. The method of claim 1,

wherein the single-location synchronizations employ respective pairs of load-linked (LL) and store-conditional (SC) operations.

19. The method of claim 1,

wherein the single-location synchronizations are compare-and-swap (CAS) synchronizations employed to define a load-linked (LL) sequence and a store-conditional (SC) sequence, respectively.

20. The method of claim 1,

wherein the load-linked and store-conditional sequences employ tagged id displacement for ABA avoidance.

21. An obstruction-free implementation of a k-compare, single-swap synchronization construct that, in an uncontended execution thereof, employs no more than two (2) atomic, single-location read-modify-write synchronizations.

22. The k-compare, single-swap synchronization implementation of claim 21,

wherein the single-location read-modify-write synchronizations employ tagged id displacement for ABA avoidance.

23. The k-compare, single-swap synchronization implementation of claim 21, further comprising:

a update mechanism that updates state of a first targeted location; and

a snapshot mechanism that verifies state of k-1 other targeted locations.

24. The k-compare, single-swap synchronization implementation of claim 23, wherein prior to a linearization point of the snapshot mechanism, the implementation displaces the first application value from the first targeted location.

25. The multi-compare, single-swap synchronization implementation of claim 24, wherein the displacing includes:

reading the first application value;

storing the read value in an auxiliary location associated with a tagged id; and

storing, using a first of the single-location read-modify-write synchronizations, the tagged id in the first targeted location.

26. The multi-compare, single-swap synchronization implementation of claim 24,

wherein the displacing is performed by a load-linked sequence that employs one of the single-location read-modify-write synchronizations.

27. The multi-compare, single-swap synchronization implementation of claim 23,

wherein the snapshotting mechanism collects (i) application values associated with each of the other targeted locations and (ii) tagged ids, if any, from corresponding tagged id locations until two successive collections indicate identical respective application values and identical respective tagged ids.

28. The multi-compare, single-swap synchronization implementation of claim 23,

wherein the implementation resets any particular targeted location, including the first targeted location and any of the k-1 other targeted locations, in connection with retrieval of an associated application value from a corresponding auxiliary location,

the resetting including displacing, using a single-location read-modify-write synchronization, a tagged id stored in the particular targeted location with the associated application value.

29. The multi-compare, single-swap synchronization implementation of claim 23,

wherein any particular application value is read either from a corresponding one of the targeted locations, if encoded therein, or from an auxiliary location associated with an id, if instead, the id is encoded therein.

30. The multi-compare, single-swap synchronization implementation of claim 23,

wherein the first and other targeted locations are non-contiguous.

31. The multi-compare, single-swap synchronization implementation of claim 21,

wherein the single-location read-modify-write synchronizations are CAS synchronizations.

32. The multi-compare, single-swap synchronization implementation of claim 21,

wherein the single-location read-modify-write synchronizations employ respective pairs of load-linked (LL) and store-conditional (SC) operations.

33. The multi-compare, single-swap synchronization implementation of claim 21,

wherein the single-location read-modify-write synchronizations are compare-and-swap (CAS) synchronizations employed to define a load-linked (LL) sequence and a store-conditional (SC) sequence, respectively.

34. A method of supporting a multiple-target synchronization operation that modifies, at most, one of the targeted locations thereof, the method comprising:

associating tagged id locations with each of the targeted locations;

displacing an application value, if any, in a to-be-modified one of the targeted locations and storing therein and in the associated tagged id location, a first tagged id distinguishable from the application value, the displaced application value being stored in an auxiliary location associated with the first tagged id;

repeatedly collecting (i) application values associated with each of the targeted locations other than the to-be-modified one and (ii) tagged ids, if any, from each of the associated tagged id locations, the collecting continuing until one of the collections indicates no change in the respective collected values and respective collected tagged ids since an immediately preceding one of the collections; and

thereafter, updating the to-be-modified one of the targeted locations only if the previously stored first tagged id remains therein.

35. The method of claim 34,

wherein the collecting of application values includes resetting the targeted locations to encode any application value displaced therefrom.

36. The method of claim 34,

wherein the displacing is performed using a load-linked sequence targeting the to-be-modified one of the target locations.

37. The method of claim 34,

wherein the atomic updated is performed using a store-conditional sequence targeting the to-be-modified one of the target locations.

38. The method of claim 34, embodied as a k-compare, single-swap (KCSS) operation executable on a multiprocessor.

39. The method of claim 34, wherein the collecting of includes:

distinguishing between application values and tagged ids stored in the targeted locations; and

retrieving the associated application value from a particular targeted location, if an application value is stored therein; and

retrieving the associated application value from the auxiliary location associated with a particular tagged id, if the particular tagged id is stored in the targeted location.

40. The method of claim 70, further comprising:

coincident with the retrieval from the auxiliary location, resetting contents of the targeted location with the retrieved application value.

41. The method of claim 70,

wherein the distinguishing includes checking for a distinguishing marker encoded integrally with an instance of an application value.

42. The method of claim 70,

wherein the method avoids an ABA problem without use of a version indication encoded integrally with an application value.

43. The method of claim 34,

wherein either or both of the displacing and the updating employ an atomic read-modify-write synchronization construct.

44. The method of claim 43,

wherein the atomic read-modify-write synchronization construct includes a compare-and-swap (CAS) operation.

45. The method of claim 43,

wherein the atomic read-modify-write synchronization construct includes a load-linked/store-conditional (LL/SC) operation pair.

46. The method of claim 34,
wherein the tagged id is distinguishes stores of a same id.

47. The method of claim 34,
wherein the tagged id corresponds to a process or thread.

48. A non-blocking synchronization construct for coordination amongst instruction sequences executable on a multiprocessor, the synchronization construct verifying contents of k target locations, $k \geq 2$, and modifying one of the locations, all in a linearizable operation.

49. The synchronization construct of claim 48,
wherein $k \geq 3$.

50. The synchronization construct of claim 48, embodied as a k-compare, single-swap (KCSS) operation executable on the multiprocessor.

51. The synchronization construct of claim 48,
wherein, in an uncontended execution thereof, the synchronization construct employs no more than the pair of single-location synchronizations.

52. A computer program product embodied in one or more computer readable media and encoding at least a portion of a synchronization operation, the product comprising:
a functional encoding of a linearizable multi-compare, single swap synchronization that updates a first application value corresponding to a first targeted location only if the application values corresponding to plural other targeted locations remain unchanged;
a functional encoding of a snapshot sequence that verifies application values corresponding to the plural other targeted locations; and

wherein the multi-compare, single swap synchronization employs a pair of single-location synchronizations to that ensure that the application value corresponding to the first targeted location remained unchanged at a linearization point of the snapshot.

53. The computer program product of claim 52,

wherein, in an uncontended execution thereof, the multi-compare, single swap synchronization employs no more than the pair of single-location synchronizations.

54. The computer program product of claim 52,

wherein the single-location synchronizations employ tagged id displacement for ABA avoidance.

55. The computer program product of claim 52,

wherein prior to a linearization point of the snapshot sequence, the multi-compare, single swap synchronization displaces the first application value from the first targeted location.

56. The computer program product of claim 55, wherein the displacing includes:

reading the first application value;

storing the read value in an auxiliary location associated with a tagged id; and

storing, using a first of the single-location synchronizations, the tagged id in the first targeted location.

57. The computer program product of claim 55,

wherein the displacing is performed by a load-linked sequence that employs one of the single-location synchronizations.

58. The computer program product of claim 52,

wherein the snapshot sequence collects (i) application values associated with each of the other targeted locations and (ii) tagged ids, if any, from corresponding tagged id locations until two successive collections

indicate identical respective application values and identical respective tagged ids.

59. The computer program product of claim 52,
wherein the multi-compare, single swap synchronization resets any particular targeted location, including the first targeted location and any of the other targeted locations, in connection with retrieval of an associated application value from a corresponding auxiliary location,
the resetting including displacing, using a single-location synchronization, a tagged id stored in the particular targeted location with the associated application value.

60. The computer program product of claim 52,
wherein any particular application value is read either from a corresponding one of the targeted locations, if encoded therein, or from an auxiliary location associated with an id, if instead, the id is encoded therein.

61. The computer program product of claim 52,
wherein the first and other targeted locations are non-contiguous.

62. The computer program product of claim 52,
wherein the single-location synchronizations are read-modify-write synchronizations.

63. The computer program product of claim 52,
wherein the single-location synchronizations are CAS synchronizations.

64. The computer program product of claim 52,
wherein the single-location synchronizations employ respective pairs of load-linked (LL) and store-conditional (SC) operations.

65. The computer program product of claim 52,

wherein the single-location synchronizations are compare-and-swap (CAS) synchronizations employed to define a load-linked (LL) sequence and a store-conditional (SC) sequence, respectively.

66. The computer program product of claim 52,
wherein the computer readable medium includes at least one medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

67. A method of supporting load-linked/store-conditional (LL/SC) synchronization on a processor that does not directly support load-linked and store-conditional operations, the method comprising:
emulating a load-linked operation that targets a location, the load-linked emulation including reading an application value associated with the targeted location, storing the read value in an auxiliary location associated with an id and storing, using a first linearizable synchronization operation, the id in the targeted location, and
emulating a store-conditional operation that targets the targeted location, the store-conditional emulation employing a second linearizable synchronization operation to ensure that the application value associated with the targeted location has not changed since success of the first linearizable synchronization operation.

68. The method of claim 67, further comprising:
distinguishing between at least two types of encodings in the targeted location to identify a proper source location for the associated application value.

69. The method of claim 68, the at least two types including:
a literal encoding of the associated application value; and
a tagged id encoding that identifies an auxiliary location in which the associated application value is encoded.

70. The method of claim 67, wherein the reading includes:

distinguishing between an application value and an id stored in the targeted location; and

retrieving the associated application value from the targeted location, if an application value is stored therein; and

retrieving the associated application value from the auxiliary location associated with the id, if the id is stored in the targeted location.

71. The method of claim 70, further comprising:

coincident with the retrieval from the auxiliary location, resetting contents of the targeted location with the retrieved application value.

72. The method of claim 67, further comprising:

selectively resetting contents of the targeted location with the retrieved application value.

73. The method of claim 70,

wherein the distinguishing includes checking for a distinguishing marker encoded integrally with an instance of an application value.

74. The method of claim 73,

wherein the application value instance encodes a pointer for aligned access to memory; and

wherein the distinguishing marker employs a low-order bit unused for the aligned access.

75. The method of claim 70,

wherein the method avoids an ABA problem without use of a version indication encoded integrally with an application value.

76. The method of claim 67,

wherein, if an id is stored in the targeted location, the reading includes resetting the targeted location to store the associated application value.

77. The method of claim 67,
wherein one or both of the first and second linearizable synchronization operations include a compare and swap (CAS) operation.

78. The method of claim 67,
wherein one or both of the first and second linearizable synchronization operations include an atomic read-modify-write operation.

79. The method of claim 67,
wherein the stored id is tagged to distinguish multiple stores of a same id.

80. The method of claim 67,
wherein the stored id corresponds to a process or thread.

81. The method of claim 67, further comprising:
creating a new id instance coincident with each execution of the load-linked operation.

82. The method of claim 67,
employed in a multithreaded computation.

83. A compare-and-swap (CAS) based, non-blocking implementation of a load-linked/ store-conditional (LL/SC) synchronization construct, the implementation including a load-linked sequence and a store-conditional sequence, wherein application values stored in a location targeted by the synchronization construct do not include a version indication encoded integrally therewith.

84. The LL/SC implementation of claim 83,
wherein, by operation of the load-linked sequence that includes a CAS operation, an application value present in a targeted location is displaced by a tagged id.

85. The LL/SC implementation of claim 84,

wherein the tagged id is distinguishable from an application value and includes a version indication encoded integrally therewith.

86. The LL/SC implementation of claim 84,
wherein the tagged id is distinguishable from a pointer value for aligned access to memory using a low-order bit position unused for the aligned access.

87. The LL/SC implementation of claim 84,
wherein the tagged id identifies an auxiliary location in which the displaced application value is stored.

88. The LL/SC implementation of claim 84,
wherein the tagged id corresponds to a process or thread that successfully completed the load-linked sequence.

89. The LL/SC implementation of claim 83,
wherein, by operation of the store-conditional sequence that includes a CAS operation, a tagged id value present in a targeted location is displaced by a tagged id.

90. The LL/SC implementation of claim 89,
wherein the application values include literal values.

91. The LL/SC implementation of claim 83,
wherein the application values include pointer values.

92. The LL/SC implementation of claim 83,
embodied as part of an application programming interface (API) that provides a callable interface to the load-linked and store-conditional sequences.

93. A computer program product embodied in one or more computer readable media and encoding at least a portion of a non-blocking implementation of a synchronization construct, the product comprising:

a functional encoding of a load-linked operation that reads an application value associated with a targeted location, stores the read value in an auxiliary location associated with an id and stores the id in the targeted location using a first linearizable synchronization operation; and

a functional encoding of a store-conditional operation that employs a second linearizable synchronization operation to ensure that the read value has not changed since success of the first linearizable synchronization operation.

94. The computer program product of claim 93, further comprising:

a functional encoding of a read sequence that distinguishes between an application value and an instance of an id and, if the targeted location encodes the id, retrieves the associated application value from the associated auxiliary location.

95. The computer program product of claim 94, further comprising:

a functional encoding of a reset sequence that distinguishes between an application value and an instance of an id and, if the targeted location encodes the id, retrieves the associated application value from the associated auxiliary location.

96. The computer program product of claim 93,

wherein one or both of the first and second linearizable synchronization operations include a compare and swap (CAS) operation.

97. The computer program product of claim 93,

wherein one or both of the first and second linearizable synchronization operations include an atomic read-modify-write operation.

98. The computer program product of claim 93,

wherein the computer readable medium includes at least one medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

99. The computer program product of claim 93,
combined with a shared memory multiprocessor that does not directly support load-linked and store-conditional instructions but rather supports an alternative atomic read-modify-write synchronization operation, the resulting combination supporting LL/SC synchronization of application code without integrally encoding a version indication with the application value.

100. The computer program product of claim 99,
combined with the application code.

101. A computer program product encoding, the encoding including:
instructions executable on a shared memory multiprocessor as a multithreaded computation that includes respective instances of load-linked and store-conditional operations; and
a non-blocking implementation of a load-linked/ store-conditional (LL/SC) synchronization construct, the implementation including load-linked and store-conditional sequences executable on the shared memory multiprocessor using one or more atomic read-modify-write instructions supported thereby,
wherein the shared memory multiprocessor does not directly support load-linked and store-conditional instructions, and
wherein application values stored in a location targeted by LL/SC synchronization construct do not include a version indication encoded integrally therewith.

102. The computer program product of claim 101,
wherein the load-linked sequence reads an application value associated with a targeted location, stores the read value in an auxiliary location associated with an id and stores, using a first one of the atomic read-modify-write instructions, the id in the targeted location, and
wherein store-conditional sequence employs a second one of the atomic read-modify-write instructions to ensure that the application value

associated with the targeted location has not changed since success of the first linearizable synchronization operation

103. An apparatus comprising:

means for emulating a load-linked operation that targets a targeted location, the load-linked emulation means including means for reading an application value associated with the targeted location, means for storing the read value in an auxiliary location associated with an id and means for storing, using a first linearizable synchronization operation, the id in the targeted location, and

means for emulating a store-conditional operation that targets the targeted location, the store-conditional emulation means employing a second linearizable synchronization operation to ensure that the application value associated with the targeted location has not changed since success of the first linearizable synchronization operation.

104. The apparatus of claim 103, further comprising:

a processor, the load-linked emulation means and the store-conditional emulation means executable thereon.

105. The apparatus of claim 103, further comprising:

processor that does not directly support load-linked and store-conditional operations.

106. The apparatus of claim 103, further comprising:

wherein either or both of the first and second linearizable synchronization operations include compare-and-swap (CAS) operations.